# 二进制漏洞挖掘系列(6)-初识整数溢出

WritenBy 東

从最早简单的栈溢出,到后来的堆溢出,格式化字符串攻击,形形色色的内存攻击手法被提出，各种方法都出现了 在 2002 年 12 月的 Phrack62 杂志上提出了(integer overflow)这样一种概念”整数溢出” 对整数溢出原理进行了详细的分析也对利用和防御上给予了说明
通常上整数溢出分为基于栈的整数溢出和基于堆的整数溢出。

## 什么是整数溢出?

整数分为无符号整数以及有符号整数两种,其中有符号整数会在最高位用 0 表示正数,用 1 表示负数 ,而无符号整数则没有这种限制。另外，我们常见的整数类型有 8 位（单字节字符、布尔类型）、16 位（短整型）、32 位（长整型）等。关于整数溢出，其实它与其它类型的溢出一样，都是将数据放入了比它本身小的存储空间中，从而出现了溢出。

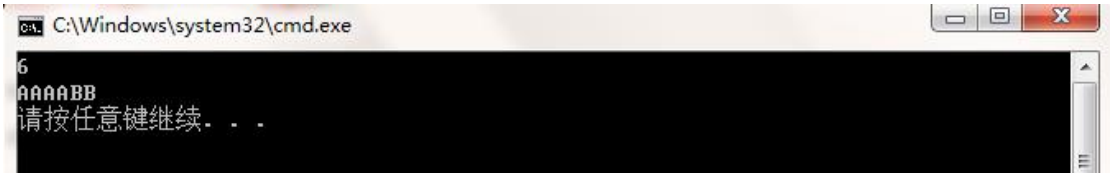
整数溢出通常分为有符号整数和无符号整数的上溢和下溢出,在利用方式上分为栈溢出和堆溢出,我们先看栈溢出的三个例子

## 1.先看一个由于无符号数不能区分负数引发的下溢

```
Unsigned char shellcode[] ="\x41\x41\x41\x41\x42\x42\x42\x42\x43\x43\x43\x43\x0f\x10\x02\x50\x65\x65\x65\x65";
int   _tmain(int argc, _TCHAR* argv[])
{
        unsigned int i;
        char str[8] = "";
        scanf_s("%d", &i);
        for (unsigned int j = 0; j <= i - 1; j++)
        {
                str[j] = shellcode[j];
                if (j >= 64)
                break;
        }
          printf("%s\n", str);
          return 0;
}
```

可以看到上面的 i 不能是大于 7 的情况如果大于 7 发生了数组访问越界,这种利用方式后面

再谈,我们看看小于 7 的时候的输出




我们在试着输入 0 试试看 全部输出了

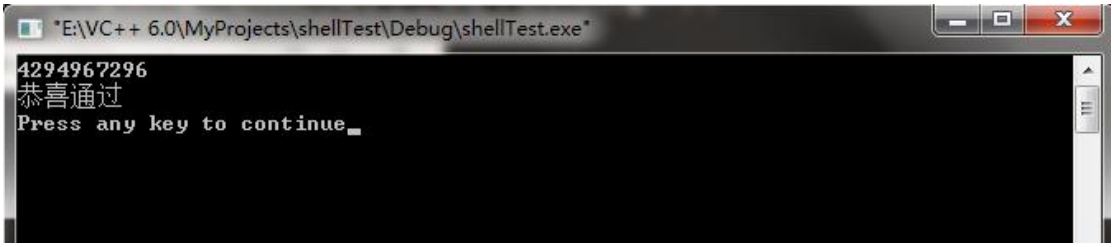

i 不大于 7 的时候输出正常,但是 i 是一个无符号数,如果我们输入 0 的话,此时 i-1 就是一个负值,j 也是一个无符号数,j<i-1 如果 i=0，那么 i-1 就是-1 所以也就绕过了数组长度的检查,由于无符号数不能识别负数,所以 i-1 转为正数 0xffffffff 很大一个数,那么就会复制超长的数据到 str,导致覆盖返回地址,这就是一个由于无符号整数下溢造成的漏洞

可以看到借此我们就可以实现任意代码执行

## 2.再来看一个无符号上溢的例子,也差不多 只不过

//在 32 位的编译器上，unsigned int 最大值:4294967295==0xffffffff

```
Unsigned char shellcode[] ="\x41\x41\x41\x41\x42\x42\x42\x42\x43\x43\x43\x43\x0f\x10\x02\x50\x65\x65\x65\x65";
int main (int argc, char *argv[])
{
    unsigned int i;
    scanf("%d",&i);
    if(i>7)
    {
        printf("输入过大\n");
        return 0;
    }
    else
    {
        printf("恭喜通过\n");
    }
    getchar();
    return 1;
}
```

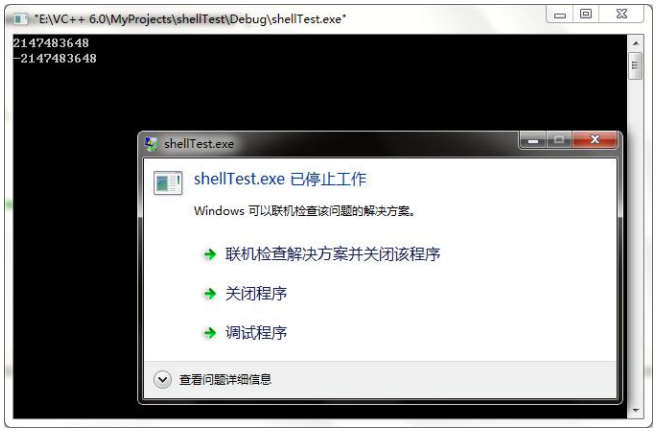



因为 4294967295=0xffffffff 所以 4294967296=0xffffffff+1=0,同样绕过了检查。

## 3.再看一个有符号数的问题

```
int main(int argc,char **argv)
{
	int len;
	char str[256];
	scanf("%d",&len);
	if(len>=256)
	{
		printf("输入有误\n");
		return 0;
	}
	printf("%d",len);
	memcpy(str,shellcode,len);  //memcpy 做内存拷贝把 len 看成无符号数使用,会溢出 str
	getchar();
	return 1;
}
```

本来是想把不大于 255 个字符由 shellcode 复制到 str，在 if(len>=256)做了检查，但是，int 是有符号数，正数的范围最大值是 2147483647。也就是 0x7fffffff。如果再加 1，它就会转为 0x80000000,十进制是-2147483648，就可以绕过 if(len>=256)检查，因为负数肯定比 256 小。然后 memcpy(str,shellcode,len);把 len 看成无符号数使用，就是 2147483648，复制这个多字符，缓冲区也就溢出了




**4.以上都是基于栈溢出的例子,再看一个基于堆的整数溢出的例子**

```
int main(int argc, char * argv)
{
	int* heap;
	unsigned short int size;
	char *pheap1, *pheap2;
	HANDLE hHeap;
	scanf("%d",&size);
	hHeap = HeapCreate(HEAP_GENERATE_EXCEPTIONS, 0x100, 0xfff);
	if (size <= 0x50)
	{
		size -= 5;
```

```
        printf("size：%d\n",size);
        pheap1 = HeapAlloc(hHeap, 0, size);
        pheap2 = HeapAlloc(hHeap, 0, 0x50);
    }
    HeapFree(hHeap, 0, pheap1);
    HeapFree(hHeap, 0, pheap2);
    return 0;
}
```


由于无符号短整数的取值范围是 0-65535,所以当 size<5 的时候 size-=5 后 size 则为负数,所以 unsigned short int 无法识别负数 从而得到正数 65534 最后分配过大堆块导致对管理结构被破坏程序从而就崩溃了